HAL
open science

Hoop: Offloading HTTP(S) POSTs from User Devices onto Residential Gateways

Kévin Huguenin, Erwan Le Merrer, Nicolas Le Scouarnec, Gilles Straub

To cite this version:

Kévin Huguenin, Erwan Le Merrer, Nicolas Le Scouarnec, Gilles Straub. Hoop: Offloading HTTP(S) POSTs from User Devices onto Residential Gateways. 21st IEEE International Conference on Web Services (ICWS), Jun 2014, Anchorage, AK, United States. pp.654-661, 10.1109/ICWS.2014.96 . hal-00965183

HAL Id: hal-00965183

https://hal.science/hal-00965183v1

Submitted on 11 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Hoop: Offloading HTTP(S) POSTs from User Devices onto Residential Gateways

Kévin Huguenin
EPFL
Lausanne, Switzerland

Erwan Le Merrer
Technicolor
Rennes, France

Nicolas Le Scouarnec
Technicolor
Rennes, France

Gilles Straub
Technicolor
Rennes, France

***Abstract*—Mobile users generate ever-increasing amounts of digital data, such as photos, which they upload, while on the go, to online services. 3G connectivity enables mobile users to upload their data while on the go but drains the battery of their devices and overloads mobile service providers. Wi-Fi data offloading overcomes the aforementioned issues for delay-tolerant data, at the cost of constrained mobility for users as they are required to stay within a given area while the data is uploaded. The up-link of the broadband connection of the access point is a bottleneck and incurs significant waiting times. In this paper, we advocate the exploitation of the storage capabilities of common devices located on the Wi-Fi access point LAN, typically residential gateways, to decrease the waiting time. We propose Hoop, a system for offloading upload tasks onto such devices. Hoop operates seamlessly on http(s) post, making it highly generic; it also requires limited changes on the gateways and on the web server and none to existing protocols or browsers. Hoop is secure and, in a typical setting, reduces the waiting time by up to a factor of 46. By correlating mobility traces with the positions of the Wi-Fi access points of a major community network, we show that Hoop drastically decreases the delay between the time a photo is taken and the time it is uploaded, compared to regular Wi-Fi offloading.**

I. Introduction

With the advent of mobile devices, users generate ever-increasing amounts of digital data while on the go. For instance, they take photos and videos with their smartphones and produce or edit possibly large documents on their tablets and laptops. The data is then uploaded to online services, typically through web applications or native apps. In many cases, the upload is performed through HTTP(S) POST operations (e.g., using a browser, or with apps relying on HTTP-based APIs).

To upload the data they produce while on the go, users rely on the connectivity of their mobile devices, namely 3G and Wi-Fi. To do so, they are offered essentially two options. Cellular connectivity enables mobile users to upload their data from anywhere (and while moving) but drains the battery of their devices [1], [2] and overloads mobile Internet service providers, which, in response, impose data caps (and either, block the traffic, reduce the bandwidth or over-charge the traffic beyond the limit) much to the detriment of the users. Data offloading at Wi-Fi access points (or 3G DropZones [3]), be they public (e.g., AT&T WiFi), business (e.g., Starbucks) or community (e.g., FON [4]) hotspots or personal or corporate access points, overcomes the aforementioned issues for delay-tolerant data. This, however, comes at the cost of constrained mobility and/or significant delays for users. Indeed, the users are required to stay in the close vicinity of the access point while the data is being uploaded. A determining factor of the upload time is the up-link speed of the Wi-Fi access point's Internet connection (typically 1 Mbps [5]) which constitutes a bottleneck compared to the Wi-Fi connection (typically 50 Mbps). The waiting time can reach ten minutes for 20 HD photos uploaded on a 1 Mbps Internet link.

In this paper, we propose to leverage on the processing and storage capabilities of common devices located on the Wi-Fi access point's local area network (LAN) to implement a sort of store-and-forward HTTP(s) proxy, thus decreasing the waiting time to the point where the Wi-Fi connection of the access point becomes the bottleneck. First-class candidates to implement such a scheme include always-on residential gateways [6], [7], routers, network-attached storage (NAS) units, and set-top boxes. One major design challenge, which is paramount for a wide adoption, is to provide a solution that is completely transparent for the users and that requires as-small-as-possible changes to existing software and protocols. We propose Hoop, a system for offloading upload tasks onto devices such as gateways in a secure and seamless way. In a nutshell, when a user reaches an HTML upload form on a Hoop-enabled website, her browser looks for a device running Hoop on the local network (say a gateway) to offload the uploading task. If such a device is found, the user's browser, instead of directly uploading the file to the online service, encrypts and uploads the file to the gateway, together with an authentication token, at a speed determined by the Wi-Fi connection of the access point. At this point, the user can disconnect from the access point, and potentially

move and switch off her device, while the file is being asynchronously uploaded by the gateway.

Hoop operates on http(s) post and relies only on existing web standards (e.g., JavaScript) and network protocols, thus making it widely applicable. More specifically, it can be used by any application that relies on http(s) post to upload data (e.g., html forms, Java uploaders, native apps). Hoop requires limited changes on the gateways and on the web server and none at the client side (i.e., at the mobile OS and browser). Hoop is secure and significantly reduces the users' waiting time.

We analyze the security of Hoop and we show that Hoop guarantees the confidentiality and the integrity of the uploaded data. In addition, we show that Hoop does not create new opportunities for an attacker to disrupt the upload or attack the online service. We evaluate the performance of Hoop in two scenarios. We consider a static user uploading data at a Hoop-enabled Wi-Fi access point and show experimentally that the waiting time is reduced by a factor of 46, compared to regular Wi-Fi data offloading. We consider a mobile user who produces and uploads data, through a major community network of hotspots, while moving and we show that Hoop significantly decreases the upload delays. We demonstrate the practicality of Hoop by implementing it on a set-top box and on a wireless router, and on various websites including a minimal html form-based uploader, the Flash and html 5 uploaders of the Gallery [8] web photo organizer, and the Java uploader of the ResourceSpace [9] web data management service.

The rest of the paper is organized as follows. In Sec. II, we survey the related work. In Sec. III, we introduce the system model and we give some background about http(s) uploads. In Sec. IV, we present and describe Hoop. We analyze the security of Hoop and we report on its performance evaluation in Sec. V. Finally, we discuss the incentives and the economics behind Hoop in Sec. VI and we conclude the paper in Sec. VII.

II. Related Work

The problem of mobile data upload has received a great deal of attention from the research community over the last few years. Balasubramanian *et al.* first proposed [10] to augment the 3G capacity in mobile scenarios by exploiting Wi-Fi access points. They implement a software solution for delaying data exchanges and fast-switching between 3G and Wi-Fi, and they assess the potential of their approach. In [1], Lee *et al.* perform a large-scale experimental evaluation of data offloading over Wi-Fi that demonstrates the benefits of this approach, both in terms of the amount of data offloaded from 3G and of battery power. In [3], Trestian *et al.* study the data generation and upload patterns of mobile users and advocate the use of cells with disproportionally upgraded bandwidth, called Drop Zones, for offloading the content generated by mobile users. In addition, they tackle the problem of the optimal placement and of the dimensioning of the Drop Zones. In all these piece of work, it is assumed that the data is offloaded directly over Wi-Fi, at the speed of the access point's connection to the Internet, which is a bottleneck. Although Hoop relies on the same approach, i.e., offloading traffic at Wi-Fi access points, it goes beyond by exploiting the storage capacity at the access points to fully take advantage of the Wi-Fi connectivity for delay-tolerant uploads.

Several pieces of work, e.g., [6], [7], advocate the use of the storage capacity of gateways–and other always-on devices–to offload data transfer from user devices. Technical solutions have been proposed and implemented on gateways, set-top boxes and networked area storage units. For instance, many such devices offer http download services and run BitTorrent clients. Closer to our work, the Fonera [11] enables users to asynchronously upload files to a number of web services (including YouTube and flickr) by simply copying them to a specific folder. Unlike Hoop, such solutions have major drawbacks: The device is trusted with the users' credentials for these web services; the device is given the users' data, in clear, which it can alter; the solution is dependent on the web service (as it relies on proprietary APIs) and it requires explicit user interactions, as opposed to Hoop.

III. System model and background

We consider a system composed of the following entities: (1) a local area network (LAN) connected to the Internet, (2) a mobile device and (3) a web service, as described in Figure 1. The local network is composed of a router (typically a gateway) that connects the different devices to the Internet, a device with processing and storage capabilities to run Hoop (typically a set-top box or the gateway), and an access point that allows users with wireless-equipped devices to connect to the local network. The user connects to the Internet (through the LAN) with her device and makes use of web services through her browser and native apps. We consider a web service that allows users to upload data through https[1] post operations, from an html form (potentially with Ajax), a Flash uploader, or a native app. Throughout the paper, we focus on the case of html forms, the other cases being in fact simpler as the service provider controls the application, whereas for html forms the service provider does not control the browser.

In a typical html scenario (without Hoop), a user connects to a web service and requests the upload page, through http(s), from her browser. The web service returns a html page including a form (e.g., see Figure 2) that contains at least a form element to select the data

[1]We focus on https throughout the paper: The case of http can be solved with a IP-level proxy; this is not possible for https, as TLS protections rely on session keys that are periodically renewed.

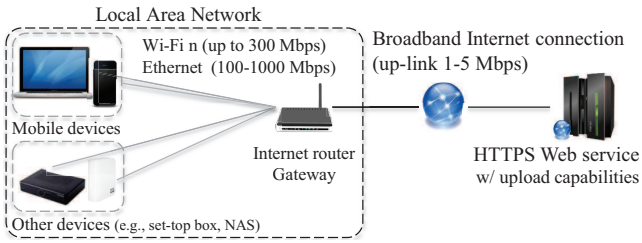


Figure 1. Setup of Hoop.

(typically some files, e.g., photos) to be uploaded, some extra information (e.g., a caption), an authentication token, and the target page (https://www.service.com/post.php) to which the data will be posted. The user then selects the file(s) to upload and submits the form by clicking on the corresponding button, and the data is posted. The user must stay connected until the data is uploaded. Once the data is uploaded, the target pages checks that the user is authenticated (e.g., based on a token stored in a hidden field of the form) and it retrieves and processes the data (e.g, adds the photos to the user's profile). The whole process is depicted in Figure 3.

```
<form id='upload_form' action='post.php' method='post'>
  <input type='file' name='data'>
  <input type='text' name='caption'>
  <input type='hidden' value='...' name='token'>
  <input type='button' value='Upload'
         id='upload_button' onclick='upload_form.submit();'>
</form>
```

Figure 2. HTML upload form.

Consider the typical scenario of a native mobile application, written in Java, for the Android platform. The application communicates with the web service through HTTP(S) in order to use the same interface as for the website: The application collects data from the local file system, as well as from various elements of the graphical user interface (GUI); the application embeds the data in an HTTP(S) request that it POSTs to the target URL by using a dedicated library (e.g., `org.apache.http.client`).

IV. Hoop

In this section, we describe Hoop, a system for offloading upload tasks onto devices located on the same LAN as the user's mobile device in a *store-and-forward* fashion. Hoop involves three different entities as described in the system model: a software component on the device running Hoop (say a gateway), the application running on the user's mobile device, and the web service. We describe the functioning of Hoop by explaining the operations performed by each of the three aforementioned entities. Hoop operates as follows: the mobile device (be it a script executed by the browser or a native app) searches for a device running Hoop on the local network and, if any such device is found, it processes (i.e., re-formats and encrypts) the data to be sent and directs the upload to this device (instead of to the web service). The device running Hoop stores the data received from the mobile device and asynchronously uploads it to the web service that handles the data as for a regular upload. We describe the general functioning of Hoop, depicted in Figure 5; and we describe the specifics of its implementation on the mobile device as a web application running in a browser and as a native app.

A. System Description

The Hoop component running at the gateway consists of a daemon acting as both an HTTP server bound to a fixed pre-defined port and an HTTP client. At the startup, the Hoop component registers the hostname `hoop.local` on the local network through the DHCP protocol. The HTTP server implemented by the Hoop component can be accessed at `http://hoop.local/` and offers two services: `test` (accessible at `http://hoop.local/test`) that allows devices to detect its presence and test its availability, and `offload` that implements the store-and-forward operation. In order to allow scripts originating from Hoop-compatible web services to connect to the gateway's HTTP services, the latter implements a cross-origin resource sharing (CORS) policy by adding `Access-Control-Allow-Origin:*` to the HTTP header.

The Hoop-compatible web service hosts, in addition to the traditional page `post.php`, a page `post_hoop.php` that handles the uploads which are offloaded to and forwarded by the gateway. Although these two pages differ in the way they retrieve and pre-process the uploaded data, they process this data in the same way by relying on the same `php` function. Thus, the modifications required at the web service are limited. The web service has a secret key K_{ws} for symmetric authenticated cryptography. Upon login, the mobile device obtains an authentication token T from the web service. When an upload operation is initiated, the mobile device obtains a fresh random secret key K for symmetric authenticated encryption, together with a version of the key encrypted with the secret key of the web service, i.e., $E_{K_{\text{ws}}}(K)$ where E denotes the encryption operation (typically AES in OCB, CCM or EAX mode), from the web service.

The mobile device (i.e., a web-application running in the browser, a Flash application, or a native app) searches for a device running Hoop on the local network by sending an HTTP request to `http://hoop.local/test`. If the request returns successfully (i.e., the host `hoop.local` is resolved and found, and the request returns the HTTP success code 200– the gateway returns the HTTP service unavailable 503 code if its upload buffer is full), the mobile device sets the target URL to `http://hoop.local/offload`, so as to offload the upload to the device running Hoop, sets

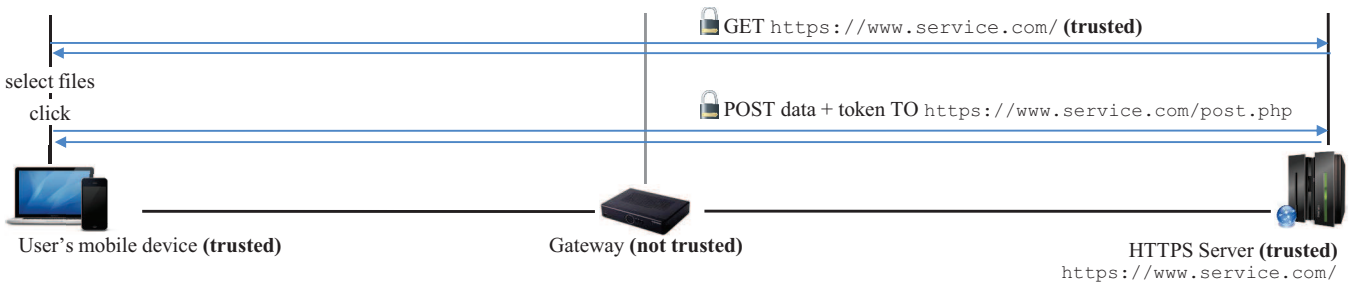


Figure 3. System overview without Hoop.

a GET parameter to the target URL of the web service (i.e., http://www.service.com/post_hoop.php), and generates the following encrypted post data: $Z = E_{K_{\text{ws}}}(K) \mid\mid E_K(T \mid\mid D)$, where T is the authentication token provided by the web service, and D is the data the user wants to upload (e.g., a photo and a caption). Note that as the content sent by the mobile device to the device running HOOP and by the device running HOOP to the web service is encrypted, there is no need to use TLS encryption (i.e., HTTP suffices); this alleviates the need for certificate management at and for the gateway. Finally, the mobile device posts the data Z to http://hoop.local/offload. As the mobile device and the device running HOOP are on the same local network, the speed at which the data is transferred is determined by the technology used on the LAN (typically 100/1000 Mbps Ethernet or Wi-Fi g/n/ac) but is independent from the speed of the Internet connection.

When the gateway receives a request to its offload service, it first extracts the target URL of the web service from the GET parameters (i.e., http://www.service.com/post_hoop.php). Then it extracts the POST data (i.e., $E_{K_{\text{ws}}}(K) \mid\mid E_K(T \mid\mid D)$) and passes this data to its HTTP client that (re-)posts it to the target URL of the web service.

The post_hoop.php page hosted by the web service parses the POST data. It first obtains the symmetric key K by decrypting $E_{K_{\text{ws}}}(K)$ with its secret key K_{ws}. Then, it decrypts the data and the authentication token by using the key K and passes them to the script used to handle regular uploads (i.e., which do not make use of HOOP). Note that when decrypting the POST data, the script checks its integrity and drops the request if it fails. Figure 4 gives a simplified version[2] of a typical post_hoop.php page (note that any language, e.g., Python, can be used for implementing post_hoop). Note that the web service relaxes its CORS policy for the post_hoop.php page by accepting any origin.

[2]For the sake of clarity, we omit error-handling code as well as diverse optimizations from the snipets.

```
function hoopReceive(){
    $fd = fopen('php://input', 'r')
    $k = hoopReadAndDecipherSessionKey($kws,$fd)
    $data = hoopDecipher($k,$fd)
    list ($_POST, $_FILE) = hoopMultipartDecode($data)
}
hoopReceive();
include("post.php");
```

Figure 4. Hoop upload php script.

B. Implementation

The implementation of HOOP as a native app on the mobile device is straightforward: sending HTTP requests is achieved by using a dedicated library such as org.apache.http.client for Java on Android; the encryption is performed by using a dedicated library such as javax.crypto. The authentication token and the encryption key are obtained from web service through HTTPS.

The implementation of HOOP as a web application however, is challenging as the web application runs within the browser over which the developer has no control. The code executed by the browser is provided by the web service as a JavaScript (over HTTPS). The script contains, in two variables, the symmetric key K and its encrypted version $E_{K_{\text{ws}}}(K)$. When the JavaScript is loaded, it searches for a device running HOOP by making an asynchronous XMLHttpRequest to http://hoop.local/test. If the request returns successfully, the JavaScript modifies the upload form in order to offload the upload to the device running HOOP. This is achieved by setting the target URL of the HTML form (i.e., its action attribute) to the empty string, and by setting instead, through the onsubmit attribute of the submit button, a JavaScript function to be executed when the form is submitted (see Figure 6). This function reads the data from the files through the HTML 5 File API, performs the encryption by using a JavaScript library[3] (e.g., crypto-js [13]), and sends the encrypted POST data to the gateway at http://hoop.local/offload

[3]While web applications usually rely on protocol-layer encryption, they increasingly use application-layer encryption: for this reason W3C is working on the specification and the implementation of a JavaScript cryptography API named WebCryptoAPI [12].

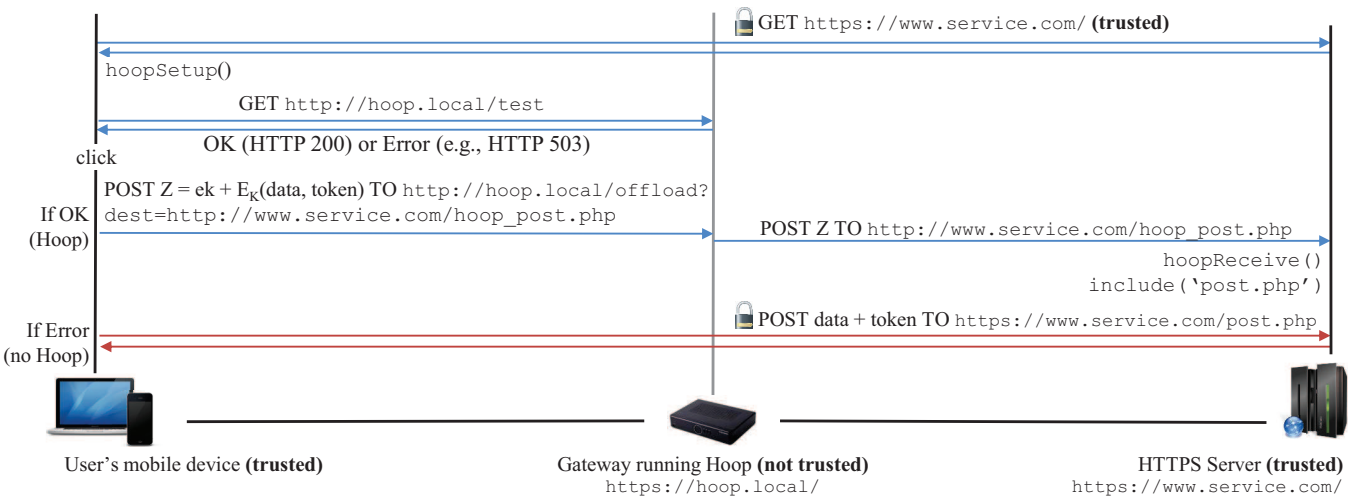


Figure 5. System overview with Hoop

by making an XMLHttpRequest with the GET parameter set to the URL of the web service (see Figure 7).

```
button = document.getElementById('upload_button');

function hoopSetup(){ // search for a device running Hoop
  req = new XMLHttpRequest();
  req.open('GET', 'http://hoop.local/test', true);
  req.onreadystatechange=function(){
    if(req.readyState==4 && req.status==200)
      button.onClick = 'hoopSend();'; // switch to Hoop offload
  }
  req.send();
}
```

Figure 6. Hoop JavaScript function for activating Hoop.

```
k = '...'; ek = '...' // symmetric key (in clear and encrypted)
dest = 'http://www.service.com/post_hoop.php'
form = document.getElementById('upload_form');

function hoopSend(){
  data   = hoopMultipartEncode(form) // extract the data
  cipher = hoopCipher(data, k)       // encrypt the data
  req = new XMLHttpRequest();
  req.open('POST', 'http://hoop.local/offload?dest=' +
       urlencode(dest), false);
  req.send(ek + cipher);
}
```

Figure 7. Hoop JavaScript function for preparing and offloading the data to a device running Hoop.

C. Additional Features

In addition to its core offload functionality, HOOP offers side features that enable users to monitor their offloaded uploads, at the gateway and at the web service. Upon a successful offload onto the gateway, the user is provided with a link of the form hoop.local/monitor?ID=..., where ID is a random identifier assigned to the offload, to monitor the (re-)posting of the uploaded data. The operator of the local network can make the monitoring service accessible from outside the LAN; in this case, the local hostname must be replaced by a fully qualified hostname. The monitoring service can be implemented at the web service as well: When an upload is offloaded to a device running HOOP, the web service is notified by the user's mobile device through an HTTPS request including the key K and the meta-data (e.g., the caption and the names of the files). The user can subsequently monitor, through her account on the web service, the list of her offloaded uploads and monitor/control (i.e., pause, resume, stop) them.

V. EVALUATION

We evaluate HOOP with respect to its security (e.g., the confidentiality and the integrity of the user's data), its efficiency (e.g., technical feasibility of HOOP), and its efficacy (e.g., in terms of its offload potential).

A. Security

We look at the security of HOOP by considering different adversaries. As HOOP is designed for a deployment in the public domain, neither the gateway nor the user is trusted, thus they constitute potential adversaries. In addition, we consider adversaries such as a jammer, an eavesdropper, or another user connected to the LAN.

Confidentiality and integrity of the users' data. The confidentiality and the integrity of the data (users' data, as well as the key K, and the JavaScript or HTML codes) exchanged directly between the user's mobile device and the web service is guaranteed by the TLS encryption of the HTTPS connection: Neither the router nor an eavesdropper can read or stealthily tamper with this data. The confidentiality and the integrity of the data exchanged between the user's mobile device and the web service, with HOOP through the gateway (over

HTTP), is guaranteed by the application-layer authenticated encryption (that implements integrity check), the encryption key K being known only to the user and to the web service as it is exchanged over HTTPS. Therefore, an adversary, such as a malicious gateway, cannot tamper with the data $(D||T)$ in the post data.

Security of the gateway. An adversary can perform a denial-of-service (DoS) attack against the gateway by issuing a large number of offloading requests. It is in general difficult to defend against DoS, yet they are not specific to the use of HOOP; this means that traditional protection mechanisms can be used and that HOOP does not create new opportunities to attack the gateway.

Security of the web service. Relaxing the CORS policy for the `hoop_post.php` page exposes the web service to cross-site scripting attacks (XSS), e.g., a third-party website stealthily posting data to this page by relying on an existing cookie in the user's browser. However, as the `hoop_post.php` page authenticates users based on the token T encrypted with the secret key K instead of using cookies, such attacks cannot succeed. Finally, an adversary can carry out a DoS attack against the web service, through HOOP, by offloading a large number of requests on a gateway that runs HOOP. Such an attack, however, does not give more power to the adversary as it is equivalent to making the requests from the LAN.

B. Efficiency

We evaluate the efficiency of the HOOP components running on the mobile device and on the gateway based on a real implementation on various platforms.

Mobile device: encryption. We considered both OpenSSL and CryptoJS libraries for symmetric AES-256 encryption and we showed (see [14]) that encryption, along with file access and communication, do not constitute a bottleneck on modern smartphones and laptops as it is possible to saturate a Wi-Fi connection at 300 Mbps.

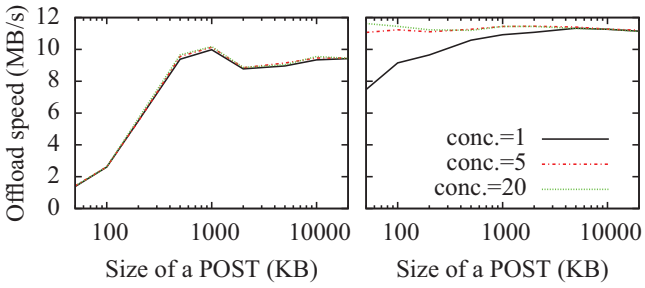
Gateway component: offload and upload. We implemented the gateway component in charge of receiving and (re-)posting offloaded data on two different devices: a wireless router running OpenWRT and a set-top box (see Table I for the configuration). The set-top box has similar hardware to a typical NAS. We implemented the HOOP component in C and compiled it to a standalone native executable. The implementation has ∼350 source lines of code (excl. the libraries) that compile to a binary of ∼60 KB (excl. a dynamic library of ∼250 KB) on both platforms. The wireless router embeds a Wi-Fi 802.11n access point and the set-top-box is connected to the router/AP through a 100 Mbps Ethernet link.

We conducted our experiments with HOOP running either on the router or on the set-top box and with a laptop connected to the local network either over 100 Mbps Ethernet or over Wi-Fi 802.11n (the actual negotiated link speed was 78 Mbps). Our experiments with a wired connection between the mobile device and the gateway enable us to assess the performance of the HOOP component running at the gateway (as a wireless connection could have constitute a bottleneck), whereas our experiments with a wireless connection enable us to assess the global performance of HOOP as a whole.

Dev.	Arch.	Proc.	RAM	HDD
Router	MIPS Atheros	AR7241@400 Mhz	32 MB	USB 320 GB
Set-top	x86 Intel	Atom@1.66 Ghz	1 GB	SATA 250 GB

Table I
SPECIFICATIONS OF THE DEVICES USED FOR THE EVALUATION.

We evaluate the performance of the HOOP component running on the gateway, in a wired setting, with respect to the offload speed (as a function of the size of the POST, for different concurrency levels[4]). It can be observed on Figure 8 that for small POSTs (e.g., 50-200 KB) offloaded onto the set-top box, sending concurrent requests improves the offload speed as the requests are processed concurrently, thus amortizing the connection delays. For large POSTs (i.e., > 1 MB), which constitute the main use-case of HOOP, both the router and the set-top box saturate the LAN connection (i.e., Ethernet at 100 Mbps∼12 MBps) at 10 and 11 MBps respectively. The bandwidth overhead of HOOP (cryptography and headers) is negligible compared to the size of the files.



(a) Offload speed (router) (b) Offload speed (set-top)

Figure 8. Performance of the Hoop component at the gateway.

C. Efficacy

We evaluate the efficacy of HOOP: first experimentally in a static setting where the users do not move and stay connected to the same access point, and then through trace-driven simulations in a mobile setting.

1) Experimental Results: We experimentally assess the global performance of HOOP in terms of the time needed to complete an offload, based on our implementation on a laptop/router (see Section V-B). This metric reflects the immediate gain of a user in a static setting, as it corresponds to the time after which the user can switch off her mobile device and/or start moving out of the

[4]Browsers can issue requests in parallel by opening up to 6-8 concurrent connections.

range of the Wi-Fi access point. For the web service, we enhance the Gallery [8] web photo organizer with HOOP compatibility[5], and we host it on a server connected to the Internet through a dedicated symmetric connection at 100 Mbps. The local network is connected to the Internet through an ADSL broadband connection synchronized at 12 Mbps (down)/1.15 Mbps (up). Neither the LAN link nor the broadband link has background traffic (i.e., other applications that use the links). Figure 9 shows the results for different POST sizes in wired and wireless settings, with and without HOOP. It can be observed that HOOP significantly outperforms regular Wi-Fi offloading (i.e., without HOOP): The offload time is reduced by up to a factor of 85 in a wired setting and by up to a factor of 46 in the wireless settings. These factors roughly correspond to the ratios between the LAN and the broadband link speeds ($100/1.15 \approx 84$ for Ethernet; the observed speed for Wi-Fi 802.11n is consistent with the actual speed of a link at 78 Mbps taking into account the MAC and TCP overheads).

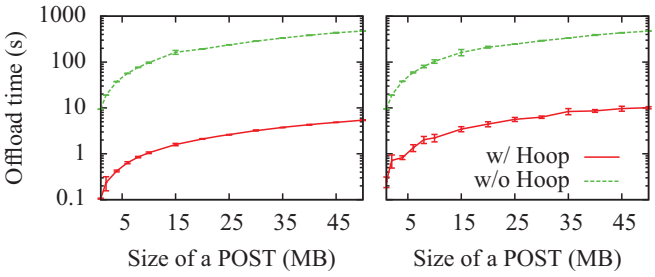


Figure 9. Global performance of HOOP compared to the baseline in a wired (left) and wireless (right) setting.

2) Trace-Driven Simulations Results: We evaluate the efficacy of HOOP in the scenario of a mobile user moving in a region covered by a network of Wi-Fi access points and a 3G network. When the user is in the range of an access point, it connects automatically to it; this is usually done by the OS (e.g., through EAP-SIM for AT&T Wi-Fi [15]) or by a dedicated app (e.g., the FON app [4]). In addition, the user could have 3G plan that enables her to connect to the Internet from anywhere. The APs have an unlimited storage capacity.

Dataset. The evaluation is based on a dataset of Wi-Fi access points from the FON community network [4]. The access points composing the FON network are mostly routers and set-top boxes provided and operated by the ISPs that hold total control over them (through automatic firmware updates); as such, they constitute first-class candidates to run HOOP. In order to build connectivity traces, we correlate the coordinates of the Wi-Fi access points with mobility traces from users moving in the Paris area, France. Our dataset comprises two types of traces: *touristic* paths and *commuter* paths. The first corresponds to pedestrians who shot photos at points of interests while exploring the city by hopping from one point of interest to another (including the Eiffel Tower, Notre-Dame, and the Arc de Triomphe); the latter corresponds to workers who edit documents while commuting between their homes and their work places using buses and trams. More details can be found in [14]. We consider the following scenarios:

- **Wi-Fi only (always mobile)**: Users always move according to their mobility trace and upload their data over Wi-Fi whenever they are connected.
- **Wi-Fi + 3G (always mobile)**: Users always move according to their mobility trace and upload their data over Wi-Fi whenever they are connected, and otherwise over 3G.
- **Wi-Fi (mobile + static)**: Users move according to their mobility traces and upload their data over Wi-Fi whenever they are connected. When connected, users wait until their upload buffer is empty before moving.

Results. Figure 10 shows the cumulative distribution functions (CDF) of the delay in the different connectivity scenarios. For the commuter trace, the CDF is not visible for the "w/o HOOP, always mobile" scenario as the delays are very long. This is because the Wi-Fi connection sessions are too short to enable the user to upload a single document. It can be observed that the delays are drastically reduced with HOOP. Surprisingly, we observe shorter delays "w/ HOOP always mobile" scenario than for the "mobile + static" (i.e., Wait) scenarios. This is because in the mobile scenario, users offload the different files in their buffers to different access points. Hence, the files are uploaded simultaneously, whereas in the wait scenario, the files are uploaded sequentially as they are all offloaded at the same access point. Finally, the results show that 3G connectivity helps reducing the delays. This is because it enables the user to upload some of the files as soon as they are produced while the user stays at a point of interest with no Wi-Fi coverage.

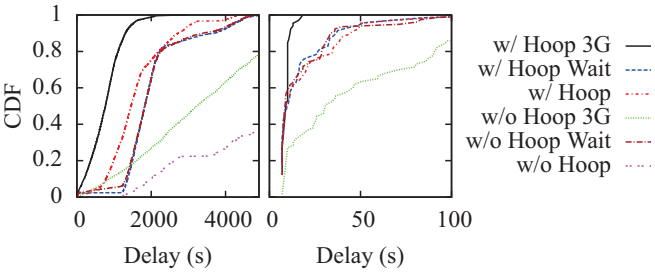


Figure 10. Delay between the content generation and the upload for the touristic (left) and the commuter (right) traces.

We now look at the active time (i.e., the time spend uploading/offloading data over Wi-Fi or 3G). Figure 11

[5]We implemented HOOP on ResourceSpace [9] as well to demonstrate HOOP's feasibility for Java-based uploaders.

summarizes the results for different scenarios: it can be observed that Hoop consistently decreases the active time while increasing the amount of data offloaded. This is consistent with the increase of the amount of data sent over Wi-Fi. This translates into energy savings as the consumption per MB is lower for Wi-Fi than for 3G [2]. Note that the results from the two traces are not directly comparable as the traces have different durations, and that the users generates files of different sizes, at different rates. Note also that for the same trace, the duration of the experiment is longer in the "Wait" scenario.

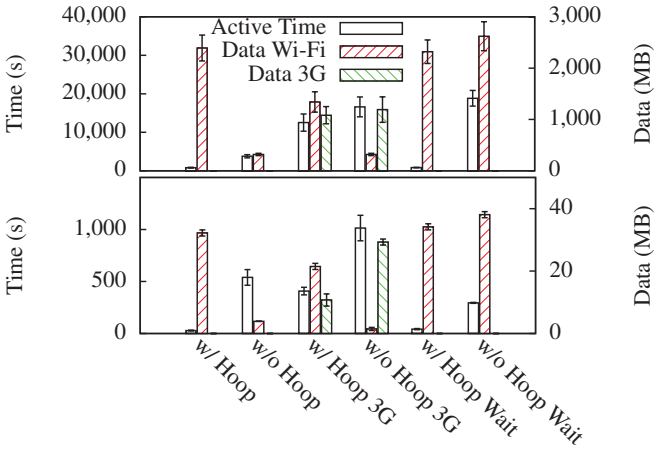


Figure 11. Time spent sending data and amount of data uploaded over Wi-Fi/3G for the tourists (top) and the commuters (bottom).

VI. Discussion

Hoop is beneficial for the users and does not require any user intervention. As such, Hoop increases the brand value (1) of gateway/NAS manufacturers, (2) of the ISPs that provide gateways to their subscribers, and (3) of Wi-Fi access point operators. The cost of deploying Hoop on such devices is minimal: The implementation is simple and can be deployed via automatic firmware updates or via applications repositories (e.g., Synology's third-party packages [16]). The fact that Hoop is generic alleviates the need for the manufacturers and application developers to implement ad-hoc solutions for each service (e.g., YouTube, flickruploaders implemented on the Fonera [11]). Hoop constitutes an interesting marketing argument for service providers as well and offers them an efficient ready-to-use solution that requires only limited changes to the web service.

VII. Conclusion

In this paper, we presented Hoop, a system for offloading data uploads on devices with storage capabilities, e.g., gateways, in a store-and-forward fashion. Our system enables mobile users to fully exploit the Wi-Fi link by relaxing the speed constraints due to the link that connects the LAN to the Internet. Unlike existing systems, Hoop operates transparently and provides a ready-to-use, secure and generic solution to data uploads offloading: The mobile users are not required to trust the gateway with their credentials and the gateway can neither see nor alter their data. We reported on our performance evaluation of Hoop, which demonstrate its efficiency and its efficacy: Hoop can run on devices with very limited capabilities and decreases the waiting time by up to a factor of 46. We intend to conduct a real-world field experiment to further assess the upload performance of Hoop as well as the energy savings.

VIII. Acknowledgments

The authors are very grateful to Olivier Heen and Julien Herzen for their insightful comments.

References

[1] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile Data Offloading: How Much Can WiFi Deliver?" *IEEE/ACM Trans. on Netw.*, vol. 21, pp. 536–550, 2013.

[2] N. Ristanovic, J.-Y. Le Boudec, A. Chaintreau, and V. Erramilli, "Energy Efficient Offloading of 3G Networks," in *MASS*, 2011, pp. 202–211.

[3] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, "Taming the mobile data deluge with drop zones," *IEEE/ACM Trans. on Netw.*, vol. 20, pp. 1010–23, 2012.

[4] "FON," http://www.fon.com, Last visited: oct. 2013.

[5] M. Dischinger, A. Haeberlen, K. P. Gummadi, , and S. Saroiu., "Characterizing Residential Broadband Networks," in *IMC*, 2007, pp. 43–56.

[6] S. Defrance, A.-M. Kermarrec, E. Le Merrer, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Efficient peer-to-peer backup services through buffering at the edge," in *P2P*, 2011, pp. 142–151.

[7] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the Internet with Nano Data Centers," in *CoNext*, 2009, pp. 37–48.

[8] "Gallery: open source web-based photo organizer," http://gallery.menalto.com, Last visited: oct. 2013.

[9] "ResourceSpace: an open source asset management," http://www.resourcespace.org/, Last visited: oct. 2013.

[10] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3G using WiFi," in *MobiSys*, 2010, pp. 209–222.

[11] "Fonera 2.0n," http://www.fon.com/en/product/fonera2nFeatures, Last visited: oct. 2013.

[12] W3C Working Draft, "Web Cryptography API," http://www.w3.org/TR/WebCryptoAPI/, jun. 2013.

[13] "CryptoJS," https://code.google.com/p/crypto-js/.

[14] K. Huguenin, E. Le Merrer, N. Le Scouarnec, and G. Straub, "Hoop: HTTP POST Offloading from User Devices onto Residential Gateways," Technicolor, Tech. Rep., 2013. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00873774

[15] "AT&T WiFi," http://www.att.com/gen/general?pid=5949, Last visited: oct. 2013.

[16] "Synology DSM 4.3 Packages," https://www.synology.com/dsm/dsm_app.php, Last visited: oct. 2013.